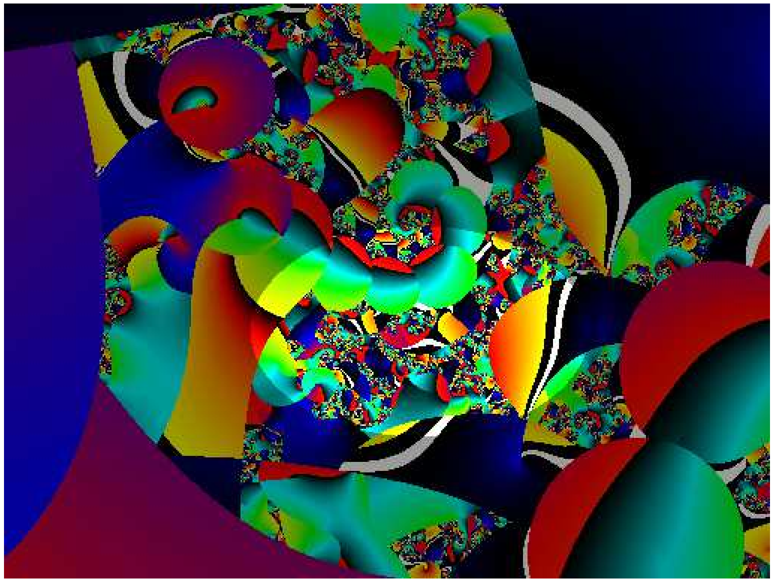
Software verification with code and models

Cyrille Artho
KTH Royal Institute of Technology, Stockholm, Sweden
`artho@kth.se`

Verification: Dynamic and Static Analysis

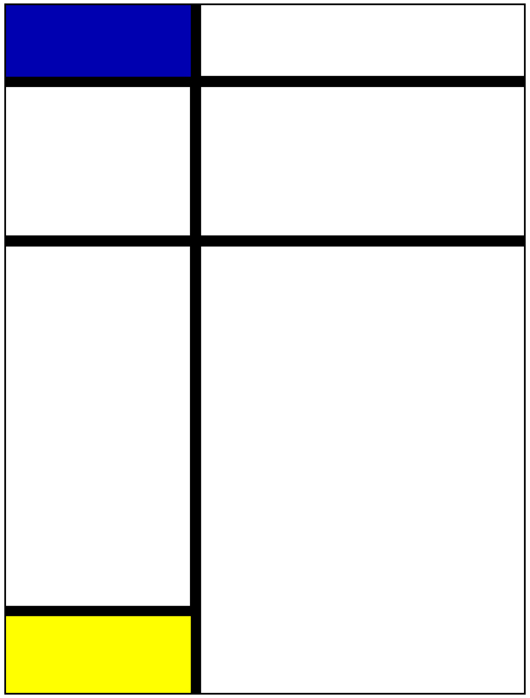
Dynamic Analysis

- “at run time”
- analyze real system



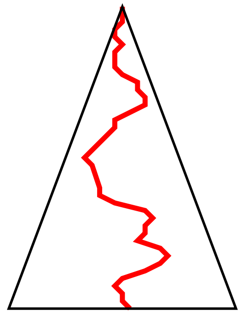
Static Analysis

- “at compile time”
- analyze simplified system



Strengths and Weaknesses

Dynamic Analysis

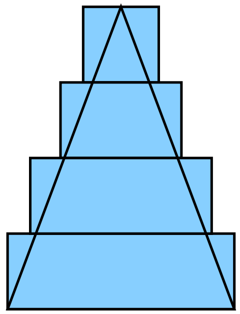


- Concrete values/states.
- Testing never exhaustive.
- May miss concurrency problems.

May miss errors

Precise information

Static Analysis



- Abstract values/states.
- Covers all possible behaviors.
- Requires precise pointer analysis.

False warnings

Exhaustive

Static analysis

- Program analysis without executing the actual program.
- Analysis of program structure/flow, independent of input.
- Good to check flow-related rules, e. g.:
  - → File (socket) must always be opened before access, closed after usage; memory must be freed.
  - → Value 0 never used in division.
  - → Array index never out of bounds.
- Tends to find a lot of „shallow” bugs, but specialized tools can also find complex problems.

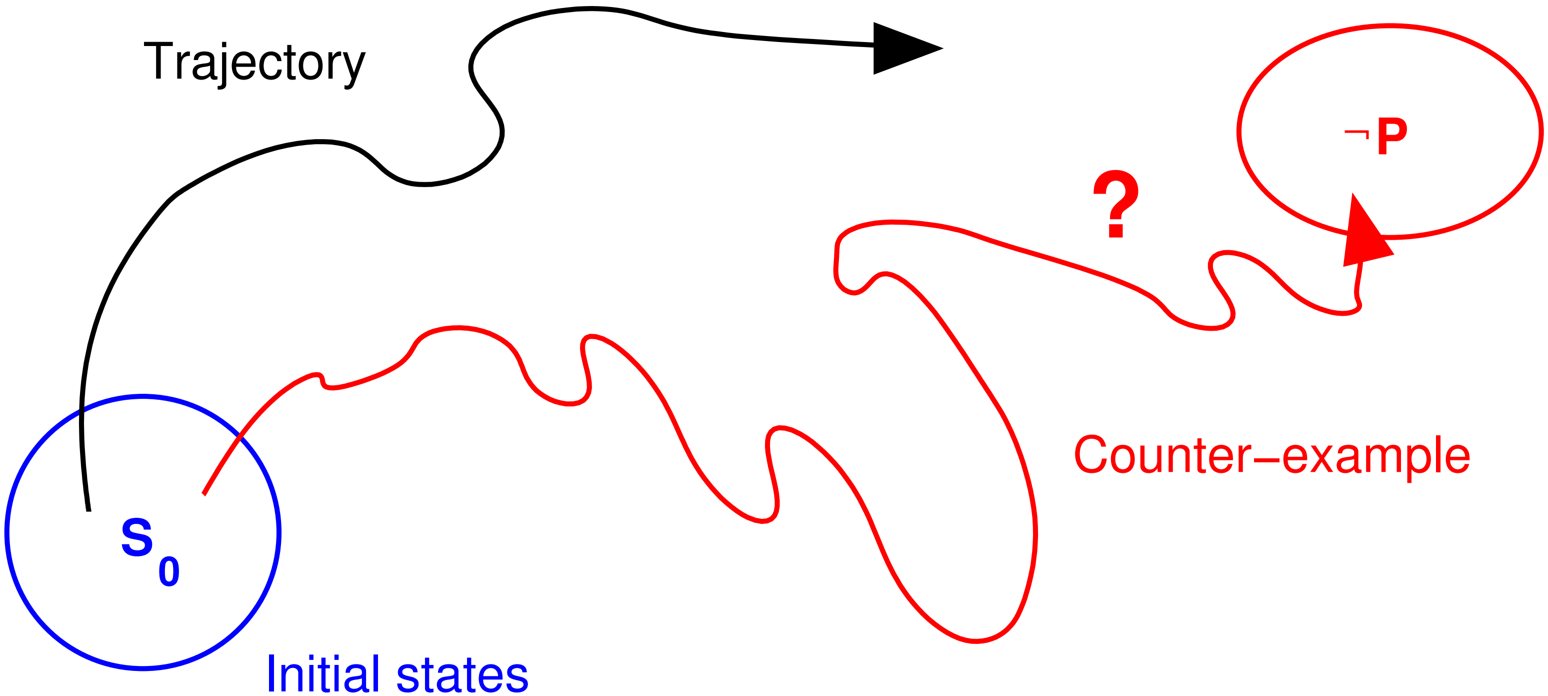
Abstract interpretation

- Abstraction: Simplify data values, e. g.,
  - $\rightarrow$ even or odd,
  - $\rightarrow$ negative, zero, or positive,
  - $\rightarrow$ within certain intervals.
- **Over-approximates** the program:
  The result represents all possible outcomes in the real program, and perhaps more.
- Example: Addition of values with sign abstraction:

$\mathrm{sign}(a)$	$\mathrm{sign}(b)$	$\mathrm{sign}(a+b)$
$-$	$-$	$-$
$-$	0	$-$
$-$	$+$	?

$\mathrm{sign}(a)$	$\mathrm{sign}(b)$	$\mathrm{sign}(a+b)$
0	0	0
0	$+$	$+$
$+$	$+$	$+$

Model checking



- Traditionally applied to specifications, protocols, algorithms.
- Certain types of software (embedded) can be mapped to such model checkers.

Theorem proving (overview)

- Mathematical program verification.
- Applicable to infinite-state, unbounded systems.
- Applies mathematical reasoning to programs, to reduce logical statements about the program to the property to be proved.
- Very complex:

  $\rightarrow$ Automated theorem provers limited in power.
  $\rightarrow$ Human-assisted tools require much human effort.

Dynamic analysis

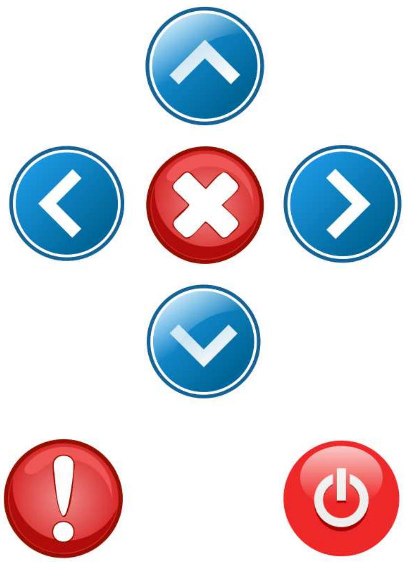
- Execution of real program on actual or simulated hardware.
- Requires test input.
- Precise analysis: outcome corresponds to real system.
- Not guaranteed to find all possible errors.
- Dynamic analysis: software testing and possible extensions: run-time monitoring, test case generation, etc.

Software testing

- Most scalable, cost-effective, and widespread verification activity.
- Test execution for given inputs can be easily automated.
- Weaknesses:
  - → May be done ad hoc (no automation, evaluation).
  - → Lack of coverage (poorly chosen inputs, not enough tests).
  - → Flaky for non-deterministic systems (concurrency, network).
  - → Test cases may add to maintenance burden.

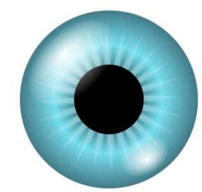
Software Testing

Input



System

Output/
observation



Can this be automated?

Unit Testing

```
@Test void test1() {
    pos = p0;
    left();
    right();
    assert(pos == p0);
}
```

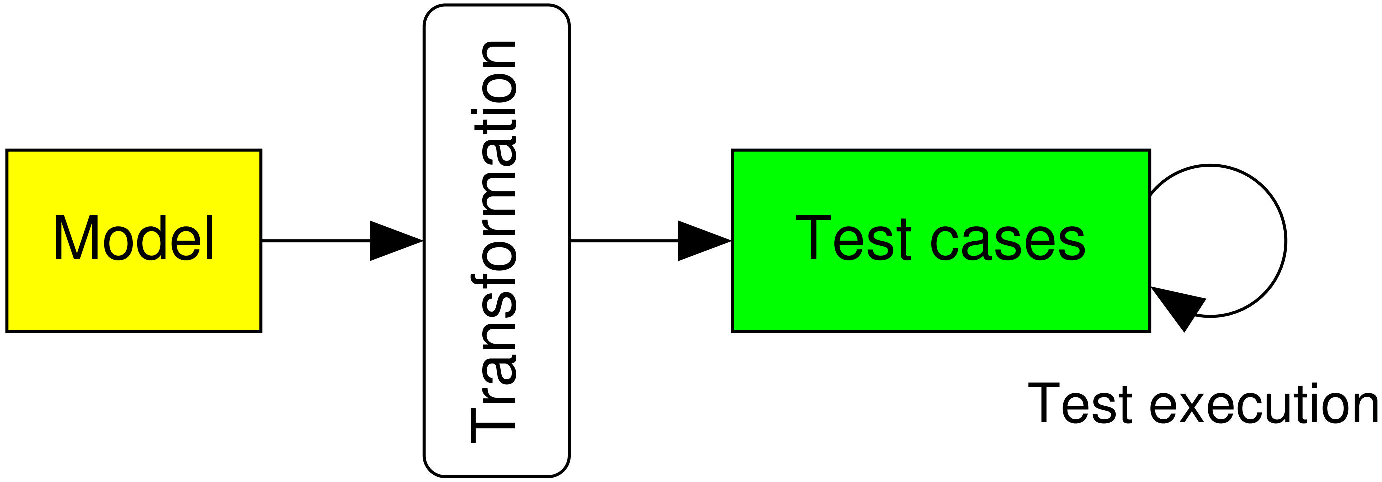
```
@Test void test2() {
    pos = p0;
    right();
    left();
    assert(pos == p0);
}
```

```
@Test void test3() {
    pos = p0;
    left();
    left();
    right();
    right();
    assert(pos == p0);
}
```

```
@Test void test4() {
    pos = p0;
    left();
    right();
    right();
    left();
    assert(pos == p0);
}
```

Can this be automated?

Model-based Testing



◆ Model contains:

→ Formalized description of the system behavior.
→ Input, expected output, exceptions, state.

◆ Transformation tool generates and executes test cases (on-line).

Different types of models/testing

Property: Find inputs satisfying preconditions, ensure postconditions.
Example: Algorithmic data structures, libraries: $x \geq 0 \cdot (\sqrt{x})^2 = x$

State-based: models behavior across multiple actions.
Example: File system, protocols.

Combinatorial: model combinations of parameters/configurations.
Example: System configurations; software product lines.

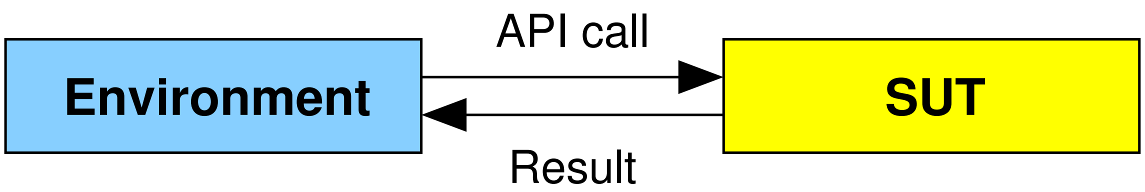
Key Challenge

Model

Real system

Model needs enough detail to create interesting test cases.

Test Model vs. System Model

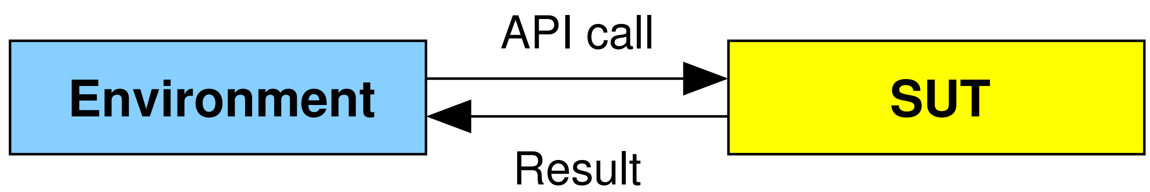


SUT = System under test; API = Application programming interface

Test model | **System model**

What | **How**

Test Model vs. System Model



SUT = System under test; API = Application programming interface

Test model

- Represents **environment**.
- Models system **behavior**.
- Used to generate **test** cases.
- Model, test one module at a time.
- Model-based testing.

System model

- Represents **system** itself.
- Models system **implementation**.
- Used to **build or verify** system.
- Need model of most components.
- Model checking, theorem proving.

Combinatorial testing (aka all-pairs testing)

1. Specify all permitted values with tables and rules.

OS	**Browser**	**Network**
Linux	Chrome	Ethernet
Mac OS	Firefox	WiFi
Windows	Safari	
	IE	

Browser = IE $\rightarrow$ OS = W
Browser = S $\rightarrow$ OS = M

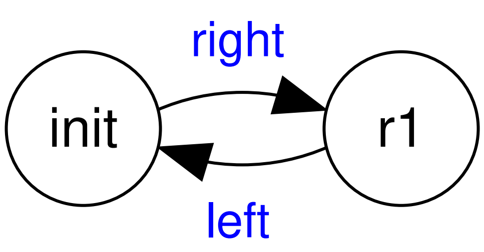
2. Generate tests to cover all *pairs* of combinations.

Test	**Covered pairs**
Linux, Chrome, Ethernet	(L, C), (L, E), (C, E)
Linux, Firefox, WiFi	(L, F), (L, W), (F, W)
Mac OS, Safari, Ethernet	(M, S), (M, E), (S, E)

Modeling state-based tests with Modbat

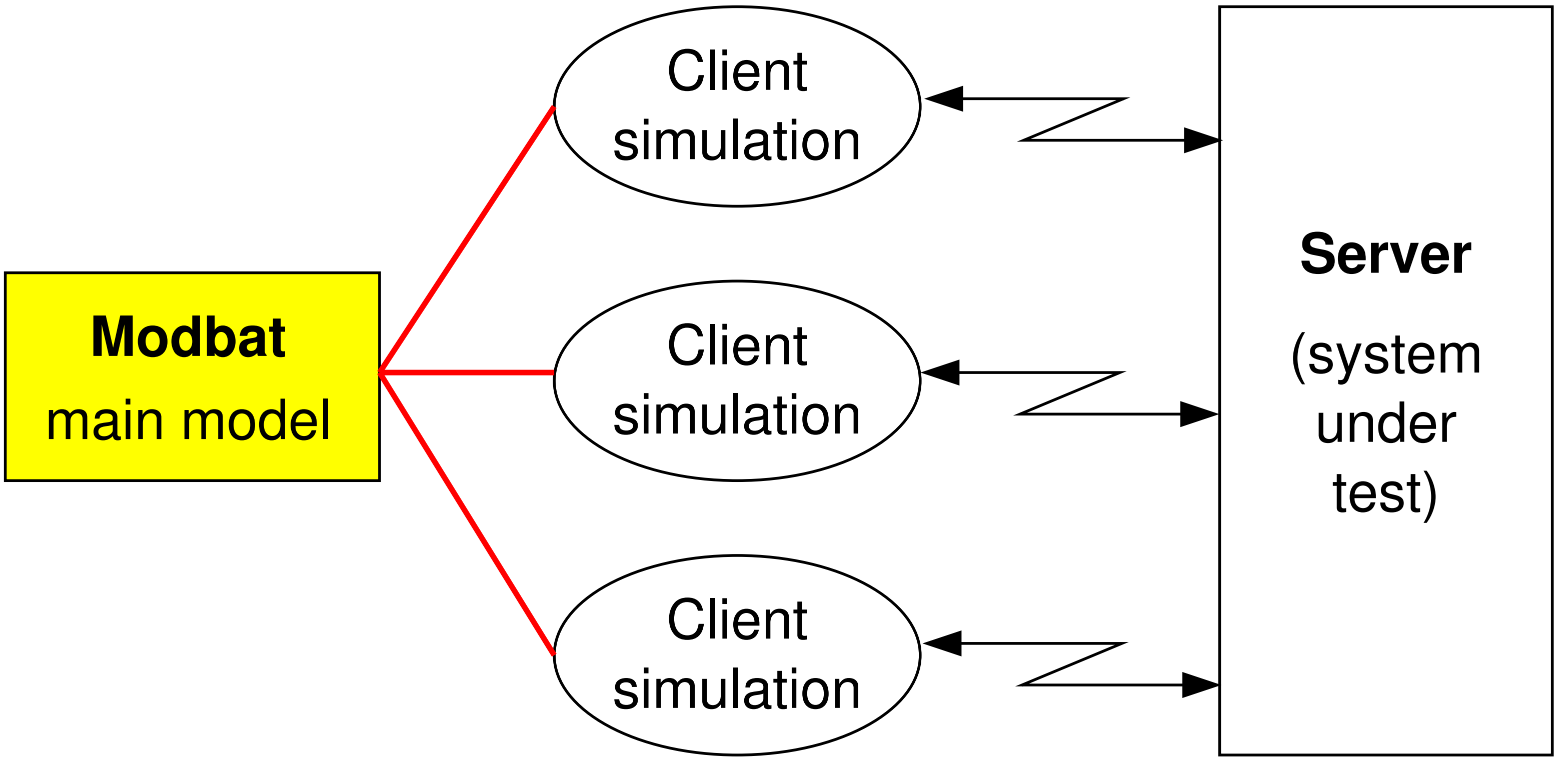
Domain-Specific Language (DSL) based on Scala.

- Extended Finite-State Machine (EFSM) as base structure.
- Add transition functions, variables for complex state.
- Structured model but flexibility of full Scala (+ Java).




```
class Example extends Model {
  var r = 0
  "init" -> "r1" := { right; r += 1 }
  "r1" -> "init" := { left; assert (r > 0) }
}
```

Combining and orchestrating test models

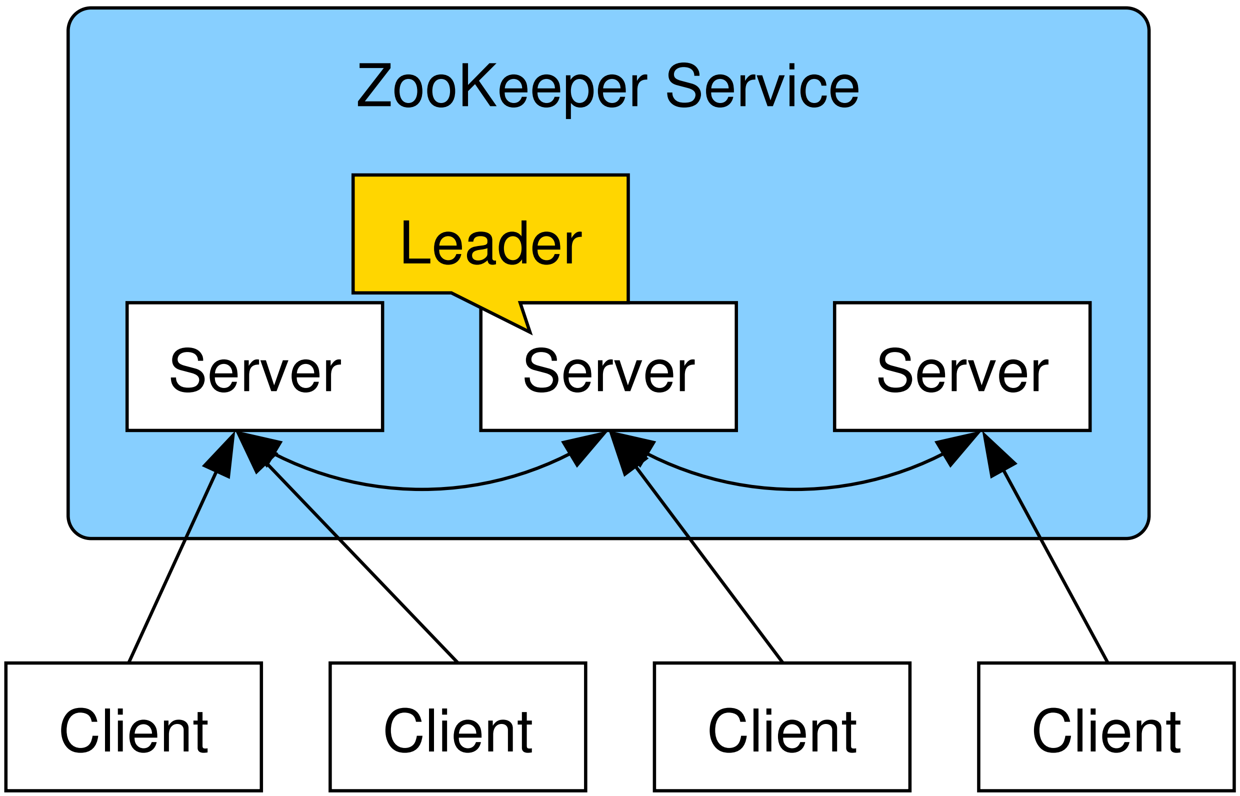


- Modbat: open-source model-based tester (@KTH).
- Main model: Test harness to start and shut down server (+ clients).
- Client simulation: Tests server functionality through client API.

Apache ZooKeeper

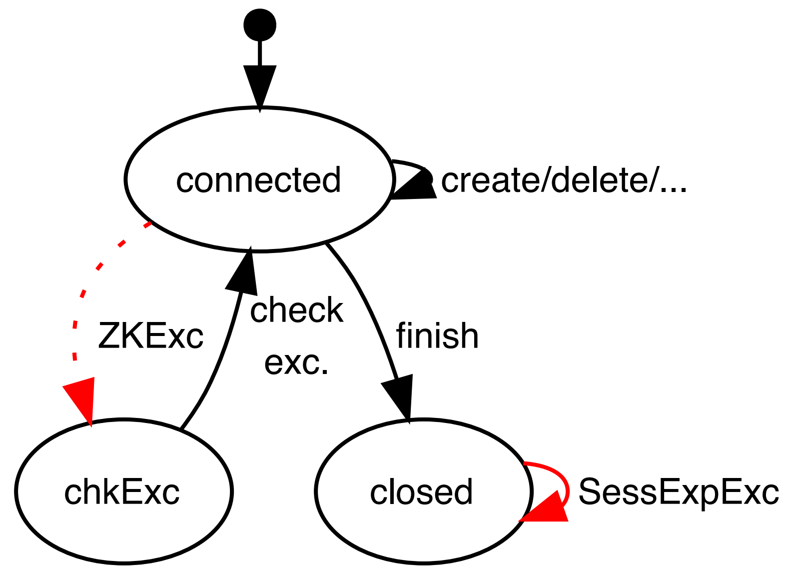
- Distributed server.
- Open source.
- Configuration, naming, distributed synchronization, group services.

Used by Ebay, Yahoo, Twitter, Rackspace, Akka, Zynga, etc.



Model-based Testing of Apache ZooKeeper

- Tested by simulating multiple client sessions.
- **Found complex defect with access permissions.**



Example tools/projects

Dynamic analysis	Static analysis
„Eraser” data race detector	„lint”-like simple style checkers
„valgrind” memory checker	„Astree”: verify Airbus software
Temporal-logic monitoring tools	seL4: proved OS kernel
„Modbat” model-based tester	KeY: interactive program verifier
PICT combinatorial tester	many protocol analysis tools

Hybrid tools, e.g., Java Pathfinder, concolic testing tools

From automated analysis to full verification

Formal proofs

Huge effort but
highest guarantee

Model-based testing
Model checking

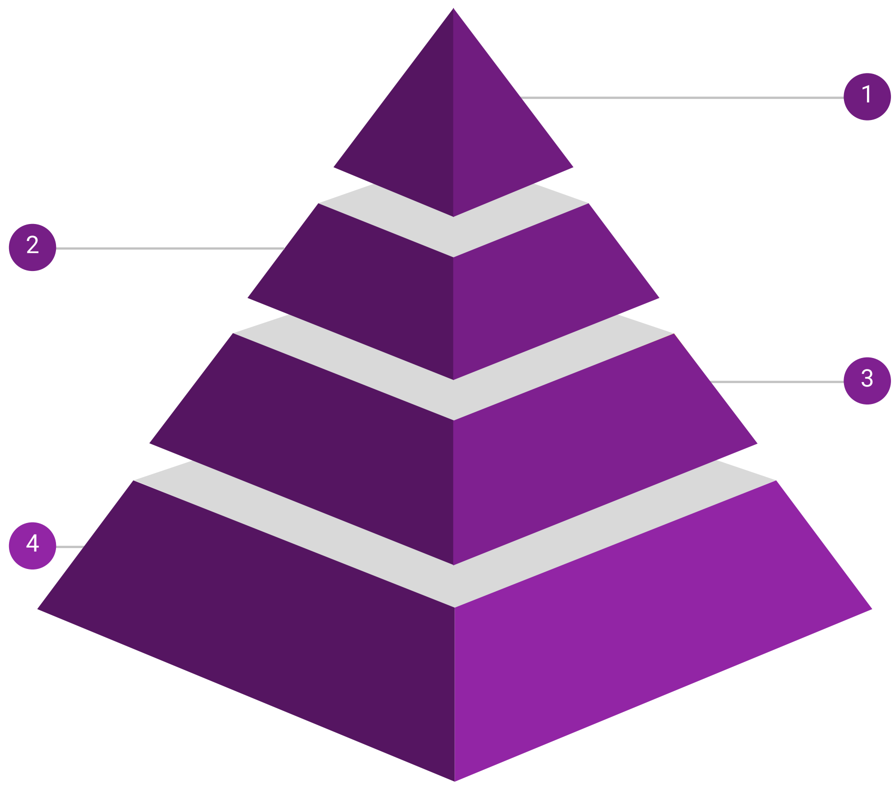
Difficult but powerful

Unit testing
System testing

Precise but limited

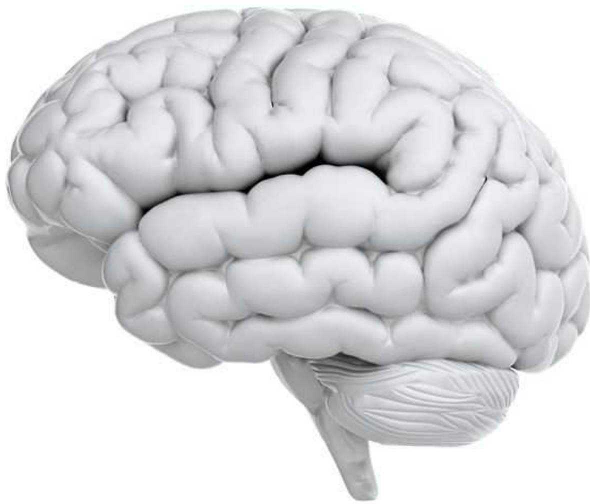
Random testing
Static analysis

Fully automated
Shallow bugs and crashes



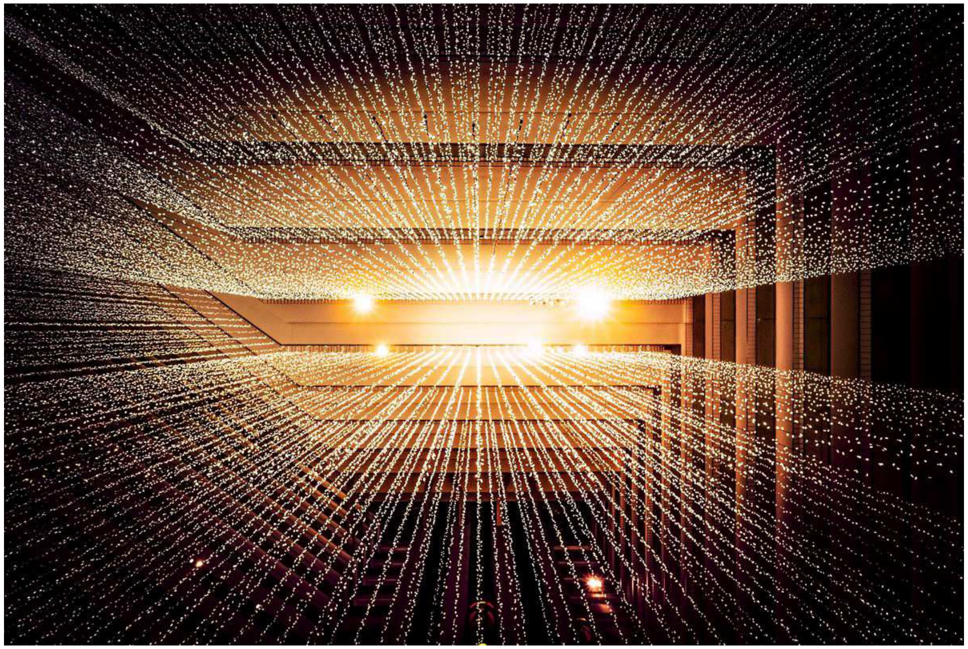
Why isn't everything automated?

Human-driven analysis



- ✔ Easy to understand, debug.
- ✘ Costly, scope often limited.

Automated analysis



- ✘ Tool output hard to read.
- ✘ Inconsistencies, not bugs.
- ✔ Automatic.

Key advantage of human insight

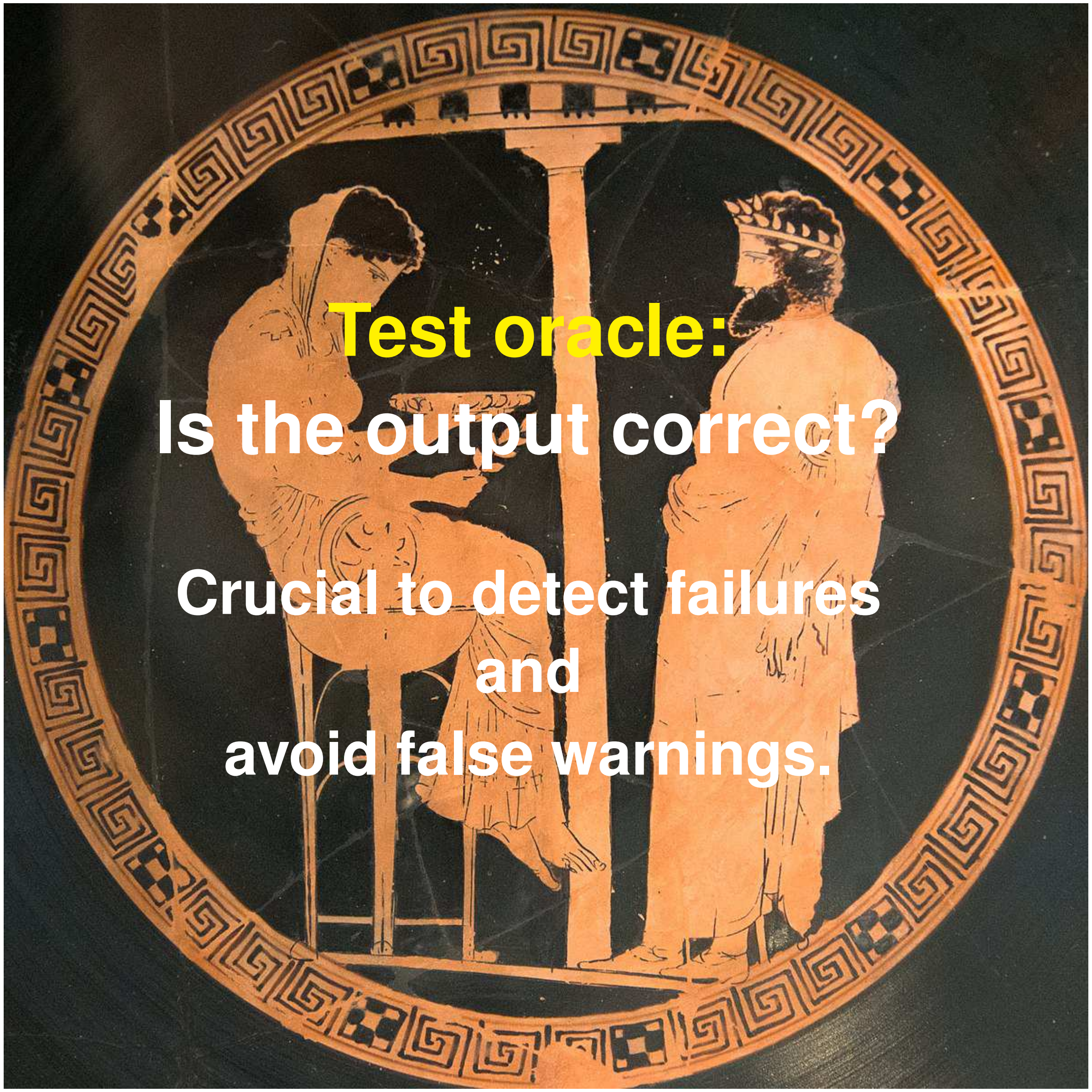
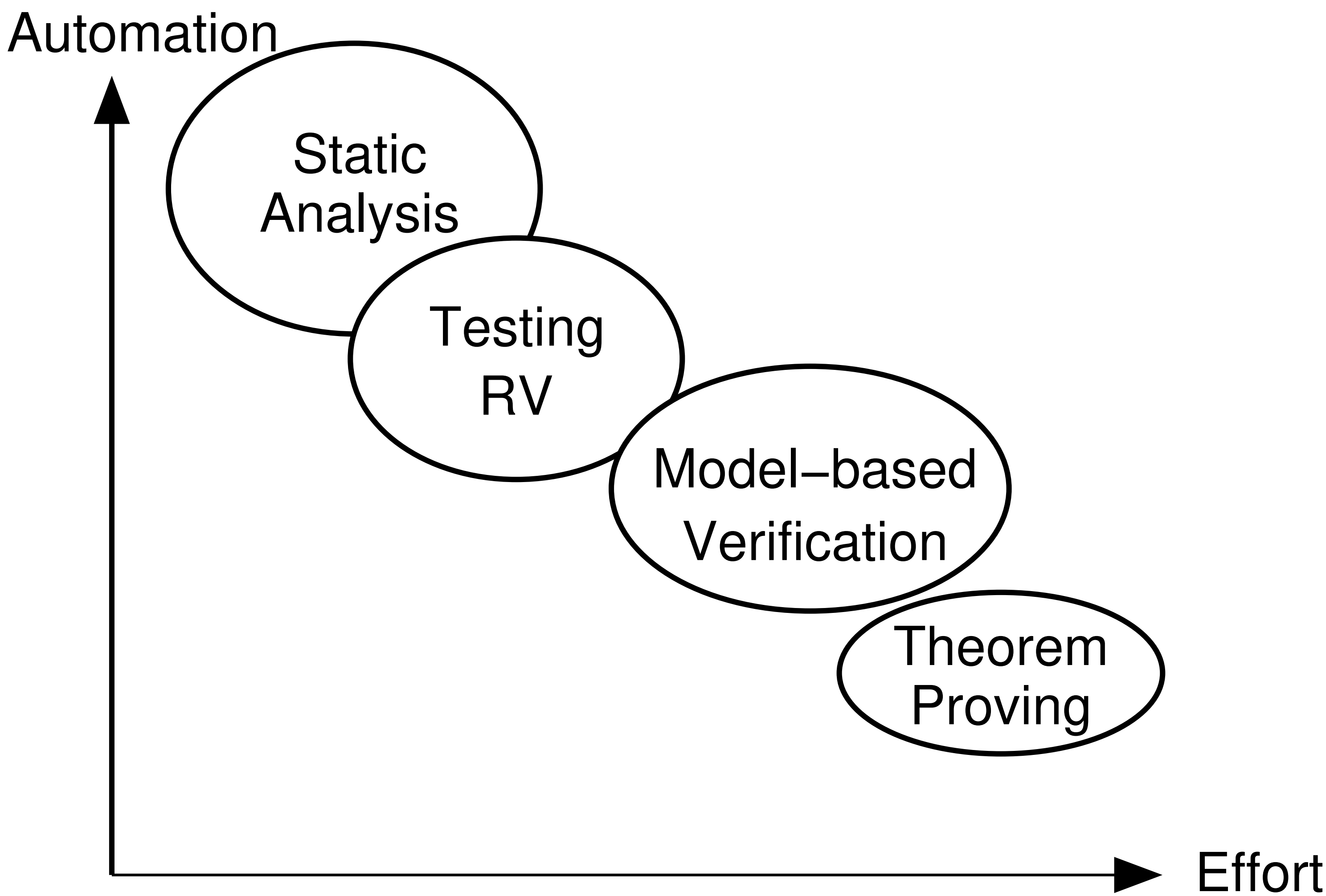
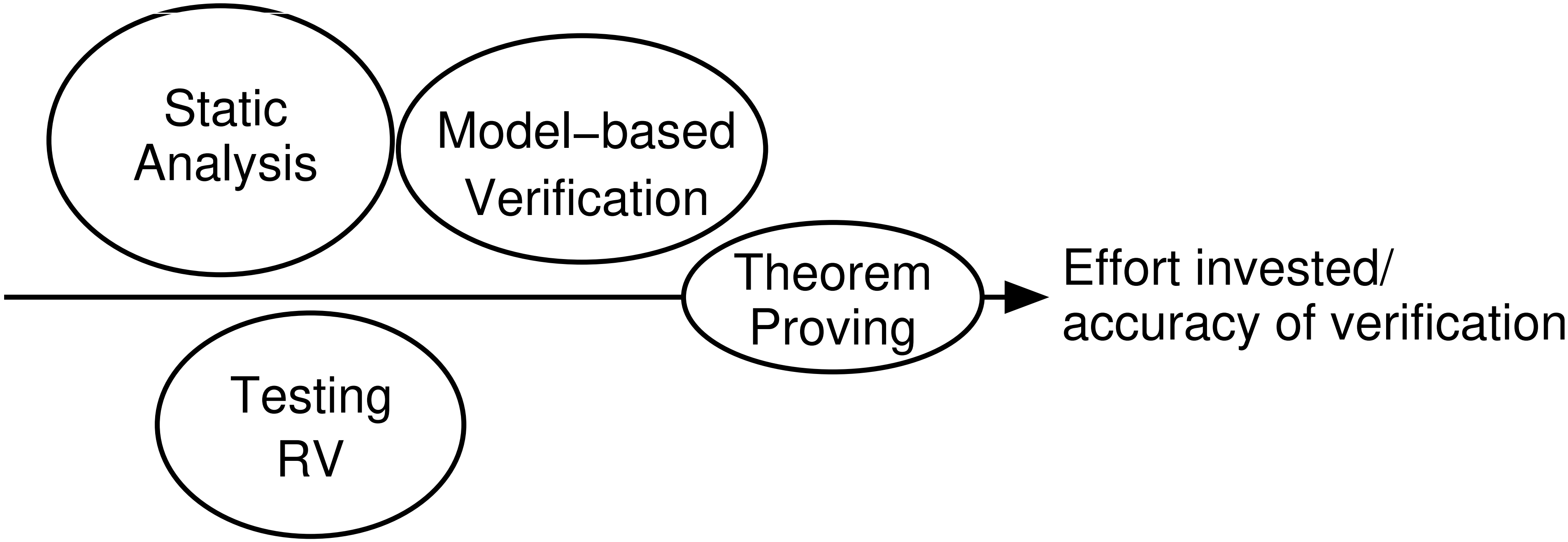


Photo by Zdenek Kratochvil on wikimedia commons

Trade-off between effort and automaton



The right tool for the job



Adapt verification to problem at hand!